CSC 317: Data Structures and Algorithm Analysis

Dilip Sarkar

Department of Computer Science
University of Miami

Outline I

Recurrence Equation for **Divide-and-Conquer** Algorithms

- With n inputs, complexity $T(n)$ of a **divide-and-conquer** algorithm
- $T(n) = f_{partition}(n) + aT(n/b) + f_{combine}(n)$, where
  - $f_{partition}(n)$ is time for dividing the problem into a subproblems,
  - a and b are integer constants
  - n/b is size of each subproblem
  - $f_{combine}(n)$ is time for combining a solutions into the solution of the original problem
- If we write $f(n) = f_{partition}(n) + f_{combine}(n)$, we get
- $T(n) = aT(n/b) + f(n)$
- Examples
  - Insertion sort: $T(n) = T(n-1) + cn$
  - Merge sort: $T(n) = 2T(n/2) + cn$
  - Quick sort: $T(n) = T(n-1) + cn$ worst-case partition
  - Strassen's algorithm for matrix multiplication: $T(n) = 7T(n/2) + \Theta(n^2)$
  - Linear-time selection: $T(n) = T(n/5) + T(\frac{7n}{10}) + c.n$
- How to solve these recurrence equations?
- We will explore three methods
  - Substitution method
  - Recursion-tree method
  - Master (theorem) method

Substitution method: Guess and Verify

- Two steps for substitution method
  - Guess the **form** of the solution
  - Use mathematical induction to find the constants and show that the solution works
  - $T(n) = 2T(\lfloor n/2 \rfloor) + n \quad (2)$
- Example:
  - Suppose, a guess is $T(n) = O(n \ln n)$
  - We need to prove that $T(n) \leq cn \lg n$ for an appropriate choice of $c > 0$ and n_0.
  - $T(n) = 2T(\lfloor n/2 \rfloor) + n$
  - $\leq 2(c\lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)) + n$
  - $\leq cn \lg(n/2) + n$
  - $= cn \lg n - cn \lg 2 + n$
  - $= cn \lg n - cn + n \; \{\lg 2 = 1\}$
  - $\leq cn \lg n \; \{\text{ for all } c \geq 1\}$
  - $T(n) \leq cn \lg n \quad (3)$
  - Now we need to verify that the solution is good.
- We have to find appropriate values for n_0 and c to satisfy the basis of the induction proof.
- $T(1) = 1$, because we have only one input
- From equation (2),
  - $T(2) = 2T(1) + 2 = 4$
  - $T(3) = 2T(1) + 3 = 5$
- But in equation (3), if we inset $n = 1$
- We get $T(1) = c1 \lg 1 = 0$, not $T(1) = 1$!
- To fix this problem, we find n_0 such that for all $n > n_0$ the equation (3) holds
- Back to equation (3)
- $T(2) = cn \lg n = c2 \lg 2 \geq 4$ for $c = 2$
- $T(3) = cn \lg n = c3 \lg 3 \geq 5$ for $c = 2$
- Thus, $T(n) = O(n \lg n)$ is correct for $c \geq 2$ and $n_0 = 2$.

Making a good guess, avoiding bad guess and pitfalls

- How do we make a good guess?
  - We will discuss one method next
- What if we make a bad guess?
  - May lead to wrong conclusion; Try with the guess $T(n) = O(n)$ for the equation $T(n) = 2T(\lfloor n/2 \rfloor) + n$
- How to avoid pitfalls?
  - We need to prove the exact form of the hypothesis
- Can change of variable help?
  - yes
- Read section 4.3 and we will discuss it in the next class

Recursion-tree is Symmetric

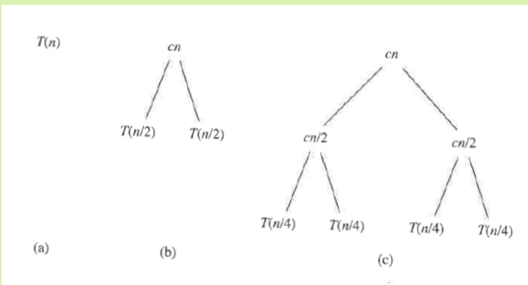
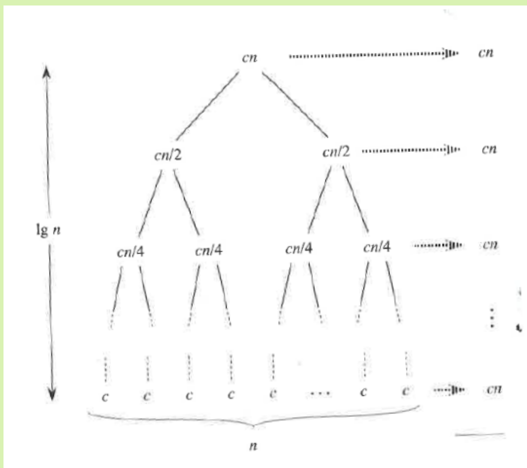


Figure: Recurrence-tree for merge sort

- $T(n) = 2T(n/2) + cn = T(n/2) + T(n/2) + cn$
- Expanding right-hand side, we get the middle tree.
- The root has cn, and two children are $T(n/2)$ and $T(n/2)$
- For the next level,
  - We expand nodes with $T(n/2)$s
- We get the tree on the right
- We continue until we have $T(1)$



- The figure above shows complete recursion tree
- What is on the right column?
  - Sum of values at each level of the tree
  - Now if we sum these values, we get an estimate of the time complexity.
- This is a **GOOD guess**
- We can use this for substitution method for solving recurrences

Recursion-tree is asymmetric

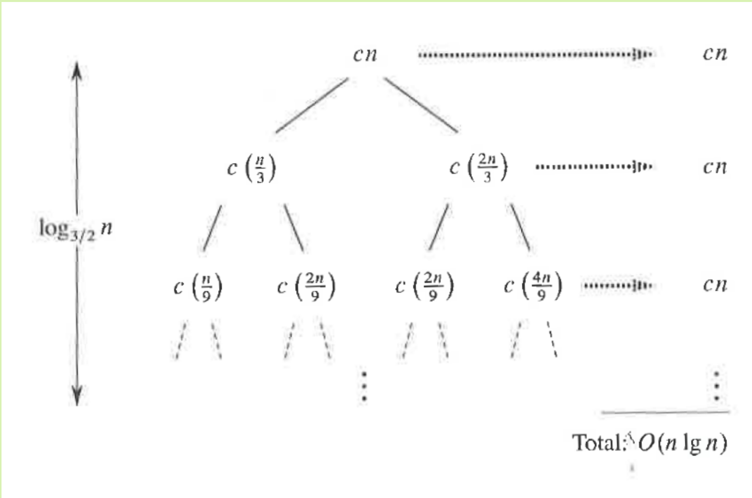


Figure: T(n) = T(n/3) + T(2n/3) + cn

- Recurrence equation,
- $T(n) = T(n/3) + T(2n/3) + cn$
- We expand as before, but
- What is the height $h(n)$ of the tree?
  - $h(n)$ is a longest path from root to a leaf node
- We get longest path following,
  - $n \rightarrow (2/3)n \rightarrow (2/3)^2 n \rightarrow \cdots \rightarrow 1.$
- $h(n) = \log_{3/2} n$
- We have to sum all the values shown on the right column.
- We get $T(n) = O(n \lg n)$

The recurrence equation for **divide-and-conquer** algorithm

- With n inputs, complexity $T(n)$ of a **divide-and-conquer** algorithm
- $T(n) = f_{partition}(n) + aT(n/b) + f_{combine}(n)$, where
  - $f_{partition}(n)$ is time for dividing the problem into a subproblems,
  - a and b are integer constants
  - n/b is size of each subproblem
  - $f_{combine}(n)$ is time for combining a solutions into the solution of the original problem
- If we write $f(n) = f_{partition}(n) + f_{combine}(n)$, we get
- $T(n) = aT(n/b) + f(n)$
- Solution of the above equation depends on how
  - $f(n)$ is asymptotically related to
  - $g(n) = n^{\log_b a}$
- Three cases to be considered
  - Case 1: For some constant $\epsilon > 0, f(n) = O(\frac{g(n)}{n^\epsilon}) \Leftrightarrow f(n) = O(n^{(\log_b a) - \epsilon})$
  - Case 2: $f(n) = \Theta(g(n)) \Leftrightarrow f(n) = \Theta(n^{\log_b a})$
  - Case 3: For some constant $\epsilon > 0, f(n) = \Omega(g(n) \times n^\epsilon) \Leftrightarrow f(n) = \Omega(n^{(\log_b a) + \epsilon})$

The master theorem

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n),$$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{((\log_b a)-\epsilon)})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{(\log_b a)})$.
2. If $f(n) = \Theta(n^{(\log_b a)})$, then $T(n) = \Theta(n^{(\log_b a)} \lg n)$.
3. If $f(n) = \Omega(n^{((\log_b a)+\epsilon)})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. □

Revisiting the master theorem

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on a the nonnegative integers by the recurrence

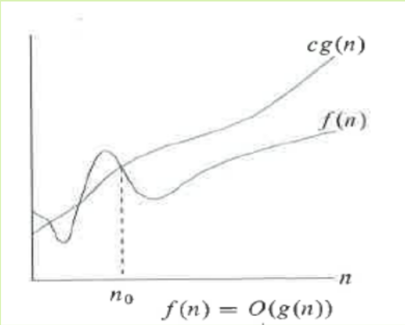
$$T(n) = aT(n/b) + f(n),$$

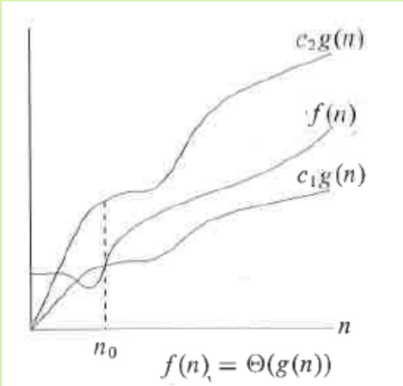
where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Let $g(n) = n^{\log_b a}$. Then $T(n)$ has the following asymptotic bounds:

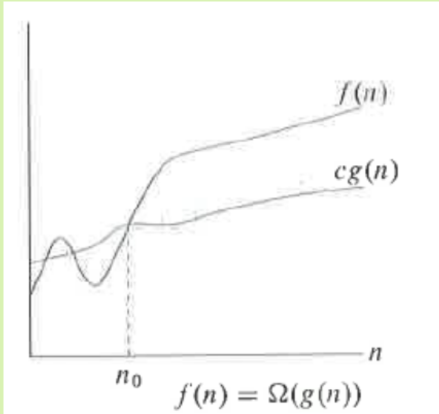
1. If $f(n) = O(n^{((\log_b a)-\epsilon)}) = O(n^{(\log_b a)}/n^{\epsilon}) = O(g(n)/n^{\epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{(\log_b a)}) = \Theta(g(n))$.
2. If $f(n) = \Theta(n^{(\log_b a)}) = \Theta(g(n))$, then $T(n) = \Theta(n^{(\log_b a)} \lg n) = \Theta(g(n) \lg n)$.
3. If $f(n) = \Omega(n^{((\log_b a)+\epsilon)}) = \Omega(n^{(\log_b a)} \cdot n^{\epsilon}) = \Omega(g(n) \cdot n^{\epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. □

Asymptotic notations and master theorem

$T(n) = aT(n/b) + f(n)$.







- Case 1: For some const. $\epsilon > 0$, $f(n) = O(\frac{g(n)}{n^\epsilon})$ $\Leftrightarrow f(n) = O(n^{(\log_b a)-\epsilon})$
- $T(n) = \Theta(g(n))$ $\Leftrightarrow T(n) = \Theta(n^{(\log_b a)})$

- Case 2: $f(n) = \Theta(g(n))$ $\Leftrightarrow f(n) = \Theta(n^{\log_b a})$
- $T(n) = \Theta(g(n) \lg n)$ $\Leftrightarrow T(n) = \Theta(n^{(\log_b a)} \lg n)$

- Case 3: For some const. $\epsilon > 0$, $f(n) = \Omega(g(n) \cdot n^\epsilon)$ $\Leftrightarrow f(n) = \Omega(n^{(\log_b a)+\epsilon})$
- $T(n) = \Theta(f(n))$

Master theorem: Case 1 Example

$T(n) = 9T(n/3) + n$

- $a = 9$, $b = 3$, and $f(n) = n$
- $g(n) = n^{\log_b a} = n^{\log_3 9} = n^2$
- $f(n) = n = n^1 = n^{(2-1)} = n^{(\log_3 9)-1}$
- $f(n) = n$ is polynomially smaller than $g(n) = n^2$
- Case 1 apply.
- Thus, $T(n) = \Theta(n^2)$.

Master theorem: Case 2 Example

$T(n) = T(2n/3) + 1$

- $a = 1$, $b = 3/2$, and $f(n) = 1$
- $g(n) = n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$
- $f(n) = 1 = n^0 = g(n) = n^{(\log_{3/2} 1)}$
- $f(n) = \Theta(g(n))$
- Case 2 apply.
- Thus, $T(n) = \Theta(g(n) \lg n) = \Theta(n^0 \lg n) = \Theta(\lg n)$.

Master theorem: Case 3 Example

$T(n) = 3T(n/4) + n \lg n$

- $a = 3$, $b = 4$, and $f(n) = n \lg n$
- $g(n) = n^{\log_b a} = n^{\log_4 3} = n^{0.793}$
- $f(n) = n \lg n = n^1 \lg n = n^{((0.793)+0.207)} \lg n = n^{((\log_4 3)+0.207)} \lg n$
- Thus, $f(n) = n \lg n$ is polynomially bigger than $g(n) = n^{0.793}$
- Hence, Case 3 apply.
- Thus, $T(n) = \Theta(f(n)) = \Theta(n \lg n)$.

What master theorem does NOT cover

$T(n) = 2T(n/2) + n \lg n$

- $a = 2$, $b = 2$, and $f(n) = n \lg n$
- $g(n) = n^{\log_b a} = n^{\log_2 2} = n^1$
- $f(n) = n \lg n = n^1 \lg n = g(n) \lg n$
- Clearly, $f(n) > g(n)$
- But we cannot apply Case 3
- $f(n)$ is NOT polynomially greater than $g(n)$
- This is evident when we compute $f(n)/g(n) = \lg n$
- There is no constant $\epsilon > 0$ for which $(f(n)/g(n))$ is asymptotically greater than n^ϵ